Rexx Objects

Dipping a toe in the object pool

Rick McGuire
2008 Rexx Symposium
An altogether too common statement:

- “these needs arise from trying not to use the oo features of oorexx since i'm creating a way for some users who know no programming language to use the minimal features of rexx.”
  - Recent comment on the REXXLA mailing list (emphasis added)
This frequently results in rejecting the easiest solution

- The discussion from the previous statement ended up as a discussion of whether interpret or value() provided the better solution.
  - did not meet *the minimal features of rexx goal*
  - ooRexx solution would have been much smaller and easier for the target users to understand
Goals of Object Rexx

Features

• Features were added with an eye toward providing easier ways to solve problems that users frequently asked about.
  – Mike Cowlishaw's “top ten” list.
  – Object orientation in many cases was the solution, not the end goal of the design.
Typical Questions

- How do I pass/return a stem to/from a procedure
- How do I expose a variable without having to expose through all call levels
- How do I drop a sub-stem
- How do I copy a sub-stem
- How do I reuse more of my code
- How do I get stem.0 to be automatically set
- How do I implement callbacks within my program
A simple example

emp.i.name = “Rick McGuire”
emp.i.location = “Sandy Hook”
....
call print_employees
....
print_employees:
procedure expose emp. empcount
do i = 1 to empcount
....
end
Common problems with using the classic approach

- The “accidental simple variable” problem.
- Writing code to deal with multiple collections.
- The external function variable scope.
- The embedded “.” problem
- Some problem solutions require use of interpret or value().
But wait...

- Structured data...
- A series of functions that operate on that data....

SOUNDS LIKE AN OBJECT TO ME!
What is an object?
Object-oriented programming is easy as...

P - Polymorphism
I - Inheritance
E - Encapsulation
A sample object

c 'SET ALT 0 0'
c 'SET DISPLAY' On On
c 'SET SCOPE DISPLAY'

c 'BOTTOM'       /* GOTOP */
c 'EXTRACT/FLSCREEN/
if flscreen.1<1 then Signal AtTop
c 'TOP'
c 'EXTRACT/FLSCREEN/
do while (flscreen.1<1)
   c 'DOWN 1'
   c 'EXTRACT/FLSCREEN/
end
Another sample object

start = 5
length = 5
data = 'Flying pigs have wings'
parse var data x1 =(start) x2 +(length) x3
Encapsulation

- “Keep your paws off my data…”
- Internal data is hidden (“Encapsulated”)
- Manipulations are only via an interface that the object defines
How do you write such a program in Rexx?

• Very difficult
  – Variable scoping rules require passing around of “globals”
  – Everything is open, everything is exposed
  – Great care must be taken for naming variables, procedures, etc., because all one shared namespace.
What is a Rexx object?

- An object is a bundle of Rexx variables ("instance variables")
- PLUS a "trusted" set of code that's allowed to directly access those variables ("methods")
- Methods may be invoked by "outsiders"
- You can have many instances of an object active at one time.
A Classic Rexx program

Main program

sub1:
procedure expose g.

sub2:
procedure expose g.

sub3:
procedure expose g.

Variables

g.
The Object picture

- Method dictionary
- The object
- Name: Rick
  Phone: 203-...
- Init
- Print
- Instance variables
  - Expose name
  - Expose name
A multiplicity of objects

method
dictionary

init

print

the object

expose name

expose name

another object

name: rick
phone: 203-...

instance
variables

name: david
phone: 607-...
A simple Rexx object

::class employee public
::method init
  expose name location
  use strict arg name, location
::method name name attribute
::method location location attribute
::method print
  say self~string
::method string
  expose name location
  return name “at” location
Creating an object

- Objects are created by sending a “new” method to a “Class” object

```
a = .employee~new(“Rick”, “Sandy Hook”)
```

- The class object allocates space, plugs in the method dictionary, and calls “INIT” to finish up construction.
Calling methods

- You call methods by “twiddling” the object

a~print
Creating your own objects

- Objects are created by making a Class object factory, and defining methods associated with the class.

```perl
::class employee
::method init
expose name address
use strict arg name, address
::method name attribute
```
The Parser...

- A real example...an object based version of the PARSE instruction
If it looks like a duck...

• ...and quacks like a duck, it's probably a duck.
Is this an XEDIT macro?

- ...or a KEDIT macro, or a THE macro?

c 'SET ALT 0 0'
c 'SET DISPLAY' On On
c 'SET SCOPE DISPLAY'
c 'BOTTOM'       /* GOTOP */
c 'EXTRACT/FLSCREEN/
if flscreen.1<1 then Signal AtTop
c 'TOP'
c 'EXTRACT/FLSCREEN/
do while (flscreen.1<1)
    c 'DOWN 1'
c 'EXTRACT/FLSCREEN/
end
Polymorphism

- “many bodies”
- In ooRexx terms, it means an object responds to the message you send it.
Pipes

- How can all of these stages work together?

'PIPE (name LIST2SRC)',
| '< fn 'listing *', /* Read the LISTING file */
| 'mtoasa', /* Machine carriage ctl => ASA */
| 'frlabel - LOC', /* Discard to start of program */
| 'drop 1', /* Drop that '- LOC' line too */
| 'tolabel - POS.ID', /* Keep only up to relocation */
| 'tolabel -SYMBOL', /* dictionary or cross-ref */
| 'tolabel 0THE FOLLOWING STATEMENTS', /* or diagnostics */
| 'outside /1/ 2', /* Drop 1st 2 lines on each pg */
| 'nlocate 5-7 /IEV/', /* Discard error messages */
| 'nlocate 41 /+', /* Discard macro expansions */
| 'nlocate 40 /', /* Discard blank lines */
| 'specs 42.80 1', /* Pick out source "card" */
| '>' fn 'assemble a fixed' /* Write new source (RECFM F) */
DO OVER

• How can DO OVER iterate over
  – An array
  – A stem
  – A stream?
• It really only understands arrays, but it sends a “MAKEARRAY” message to the object to get one.
• Any object can provide a MAKEARRAY method and work with DO OVER.
Never write this program again

select
  when type = 1 then call printEmployee
  when type = 2 then call printManager
  when type = 3 then call printExecutive
  when type = 4 then call printContractor
end
...do this instead

anEmployee~print
The TreeTable

- The tree table is polymorphic with the ooRexx Directory class
- A totally new implementation
  - Can be used interchangeably with directory objects
Standing on the shoulders of giants...

• One of the major benefits of O-O programming is code reuse
  – Don't copy the code and modify...
  – Use the original directly and extend and override.
Inheritance

• When you create a class, you can start by “subclassing” an existing class.
• You “inherit” the methods and data of the existing class...
• ...and add some of your own.
Why *inherit*?

- Extend existing function
- Alter/extend the behavior of an existing class to meet your requirements
- Complete the implementation of an abstract concept (inherit from a “framework”)
- Another means of achieving polymorphism
Enhancing the function

• Add additional capability to an existing class
  – Q: How hard would it be to add regular expression support to the PARSE instruction yourself?
  – Q: How hard would it be to add regular expression support to the Parser sample yourself?
The enhanced parser

- Same base parser, but additional function added
Getting a little SELFish

- In any ooRexx method, the variable SELF will point to the object you use to invoke the method.
  - This allows you to invoke “subroutines” using your own context:

```::method string
return self~name “living at” self~address```
Before, after, and in between

- When you subclass, you can override methods of the superclass, but still use those methods

```::method string
return "This is my version of" self~string:super```
Making callbacks

• Some classes define empty methods and allow you to fill in the blanks:

```::class myparser subclass xmlparser
::method start_element
use arg chunk
call charout, '<chunk-tag'
if chunk-attr <> .nil then do f over chunk-attr
  call charout, "F="self-translate(chunk-attr[f])"
end
say '>
return
::method end_element
use arg chunk
say '</chunk-tag>'
return
::method passthrough
use arg chunk
say '<chunk-text>'
return```
All we are saying, is give peace a chance...

- Allow the ooRexxx language to help you with what you're already trying to do!
- Using ooRexxx features doesn't require a complete reshaping of your mind set...
  - immediately rejecting these features frequently means you're working too hard!
Object-oriented programming is easy as...

P  olymorphism
I  nheritance
E  ncapsulation